[20/08/24]

Java

⇒ What is Java?

⇒ Programming language which is used to develope the software/ applications is called as "Java"

⇒ Why it is so popular?

⇒ Because of its feature it is popular.

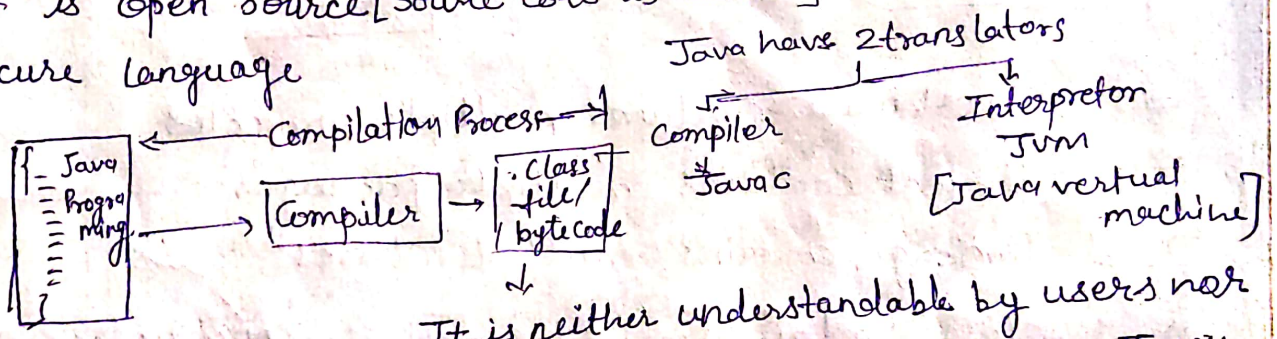
Features of Java

⇒ It is simple language [easy to understand]

⇒ It is free [No need to pay single rupee]

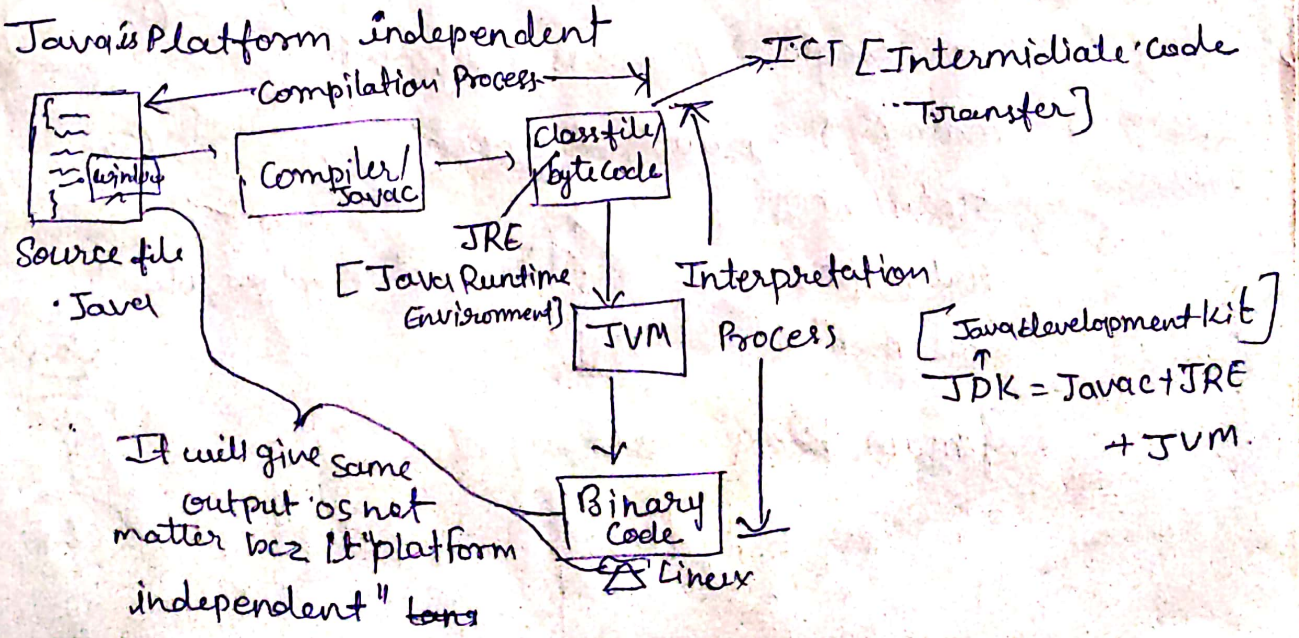
⇒ It is open source [source code is visible]

⇒ Secure language



It is neither understandable by users nor by system. excepet JVM hence Java is secure language

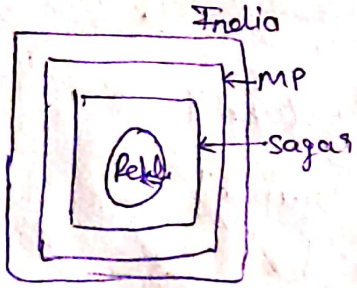
⇒ Java is Platform independent

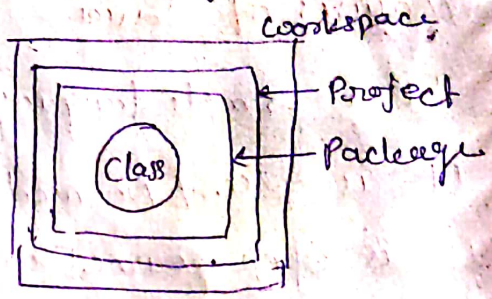


The programe is executed in one OS can easily executes in other operating systems is called as platform independent nature of Java / Architectural neutral Behaviour of Java.

NOTE - To run Java programms we should have to Install Eclips plateform.

How can we track our location ⟺ where we can write our Java program.





⇒ 1 workspace contains multiple projectes

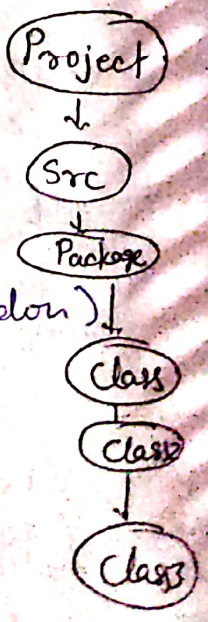
⇒ 1 Project contains multiple packages

⇒ 1 Package contains multiple classes.

⇒ 1 class contains Variables, methods, constructors, Conditional statements, Control statements, bloks [STB, ITB].

Structure of Java program :-

```
Class Demo {// class body open
main (-) { main body open
statement 1 . ---- ;
statement 2 . ---- ; } separator (semi colon)
statement 3 ------ ;
}//main body close
}// class body close.
```



Scanned with OKEN Scanner

Printing Statement In Java:-

System.out.println("Hello");
↓
It will print data cursor immidiatly goes to next line

System.out.print(" ");
↓
It will prints the data cursor will not jumps next line and same line it will print

Short cuts

⇒ for main method = main (control + space)
→ For printing statement - syso (control + space)

[21/08/24]

Variables:- Small piece of memory in which user can store single data at a time is called as "Variable."

Datatypes:- It is used to represent which type of data we are going to store inside the variables.

There are two types of Datatype -

Primitive Data type (PDT)

⇒ 8 PDT's are there
⇒ Memory size is fixed
⇒ All PDT's are "keywords"

non-primitive (PDT)

⇒ Infinite PDT's are there
⇒ Memory size is not fixed
⇒ All non-PDT's are identifires.

Primitive Datatypes		Size
byte	→	1 byte
short	→	2 byte
int	→	4 byte
long	→	8 byte
double	→	8 byte
float	→	4 byte
char	→	2 byte
boolean	→	1 bit

byte is a unit of memory

1 byte = 8 bits

1 0 0 0 1 1 1 1 ⇒ 1 byte
8 7 6 5 4 3 2 1
single bit

Scanned with OKEN Scanner

[22/08/24]

Variables :-

datatype → int i ; → variable name
separator

Variable Declration ;
Variable Initialization

OR

int i ;
i = 10 ;

int i = 10 ; ⇒ Variable Declration and Variable Initialization within single statement

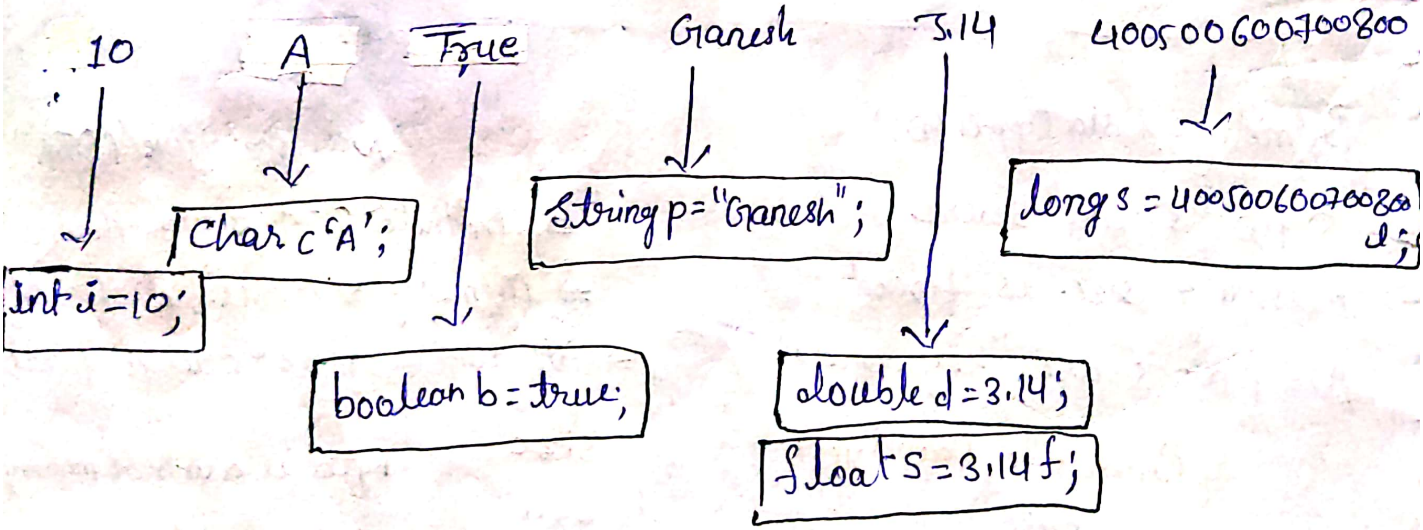
Integers
10, 20, 30 - - - - -
byte b = 10 ;
short s = 20 ;
int i = 55 ;
long d = 6224566070301 ;

floating Decimals
3.14, 10.5
float a = 3.14f ;
double b = 10.5 ;

characters
A, B, C, Z
Char c = 'A' ;
Char b = 'Z' ;

boolean datas
true / false
boolean b = true ;

Strings
" "
String s = "Testing" ;
String d = "10" ;



Scanned with OKEN Scanner

Variables

Local Variables

The variables which are declared inside the main method body and class body

```
Class A {
main(--){
    int i = 10;
}
}
```

Global variables

The variables which are declared inside the class body and outside the method body

```
class A{
    int i=20;
main(--)
 {
 }
}
```

[23/08/24]

Rules for Local Variables :-

⇒ Can't print local variables without initializing it

ex - int i; // VD → i □

syso(i); // → nothing will print.

⇒ We are printing variable means we are printing data inside the variable

int j = 20; → j 20

SYSO(J); // 20 - will print

⇒ Duplicate local variables are not allowed

int i = 10; —— JVM will get confused

int i = 20;

⇒ No need of any access modifier for local variables.

public, private [scope is always inside the method

Scanned with OKEN Scanner

⇒ We can reinitialize local variable any no of time but previous values can be deleted and latest values can be updated.

ex - int k = 10; → k [10] ×

k = 20; → k [20] ×

k = 30; → k [30] — will print

Syso (k); — 30 will print

Rule for Global Variable

⇒ We can print global variable without initializing it By default compiler will gives values

byte, short, int, long = 0

double, float = 0.0

boolean = false

string = null

char = □ → empty char box

```
class A {
static int i;
main (---) {
  Syso (i); // 0 will print
```

NOTE :- Compiler will provides default values only for static Global variables.

Static → single copy

non static → multiple copies [xerox]

Scanned with OKEN Scanner

Assignment :- Create a Java class and Take 2 local variables and 2 static Global variables and Print the value of local variable?

Java class for 2 local variables :-

```
Package Assignment
Public class LVA{
    Public static void main (String[] args){
    Char c ='A';
    String j = "Pratha";

System.out.println(c);
System.out.println(j);

    }
    }
```

Java class for 2 static Global variable

```
Package Assignment
Public class GVA {
    static boolean d = true;
    static double k = 20.5;
Public static void main (String [ ] args) {
    System.out.println (d);
    System.out.println (k);
}
}
```

Scanned with OKEN Scanner

⇒ Create a class .try to print non-static global valiable in main method ?

Package Assignment

```
Public class B {
        int l;
        int h;
        int t;
    Public static void main (String [] args) {
        B e1 = new B();
        B e2 = new B();
        B e3 = new B();
    System.out.println("value of non-static:"+e1.l);//0
    System.out.println("value of non-static:"+e2.h);//0
    System.out.println("value of non-static:"+e3.t);//0
    }
}
```

[0 will print]

Scanned with OKEN Scanner

[24/08/24]

Static

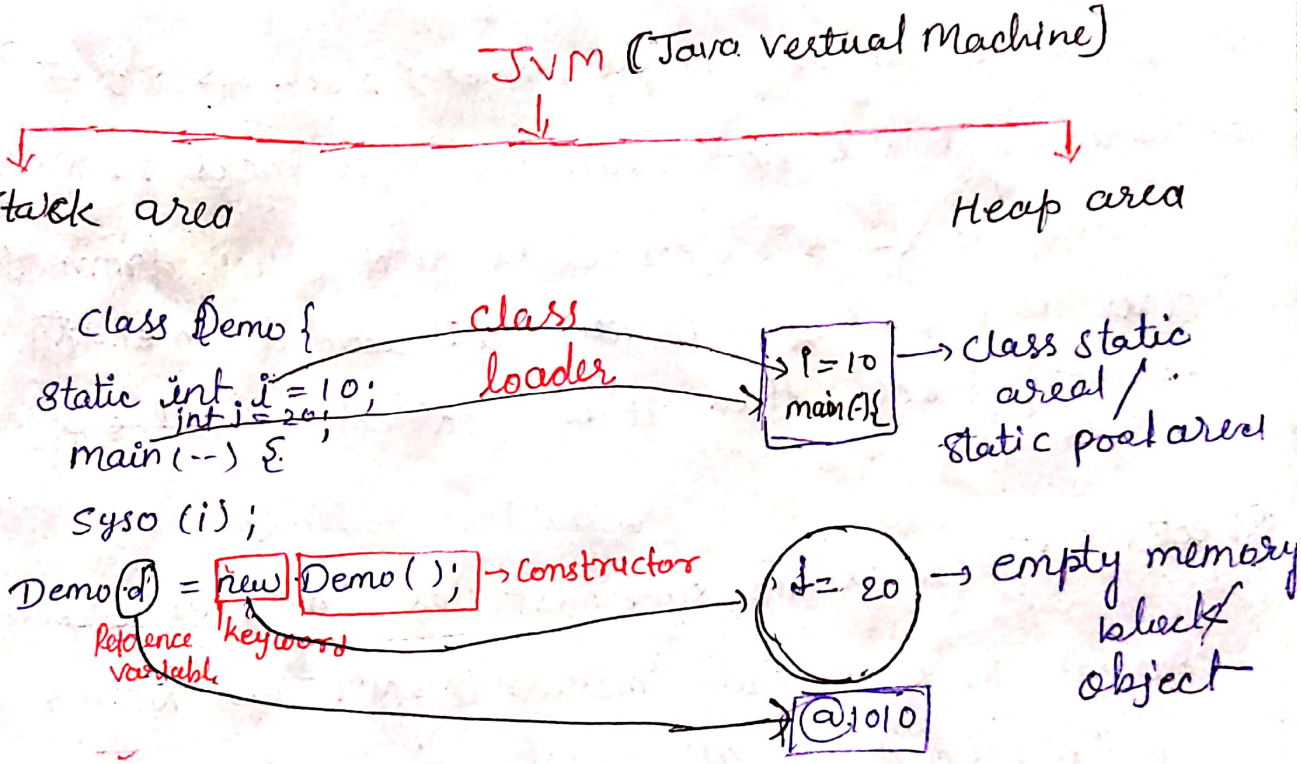
⇒ It is having single copy

⇒ These are loaded in class static area by class loader

Non-Static

⇒ It is having multiple copies

⇒ These are loaded in object inside the heap area which is created by new keyword by constructor.



⇒ The empty memory block which is created by new keyword inside the Heap area is called "Object."

Object = Demod = new Demo (); → mandatory to run non-static global variable

JVM → It is having 2 areas stack & Heap. Whatever program we are writing that can be stored in stack area and their instance/replicas are generated under Heap area.

Scanned with OKEN Scanner

Class loader :- It will loads static members into class static area.

Constructor :- It will loads non-static members into objects which is created by new keyword.

New keyword - 1. It will creates empty memory block in heap area.

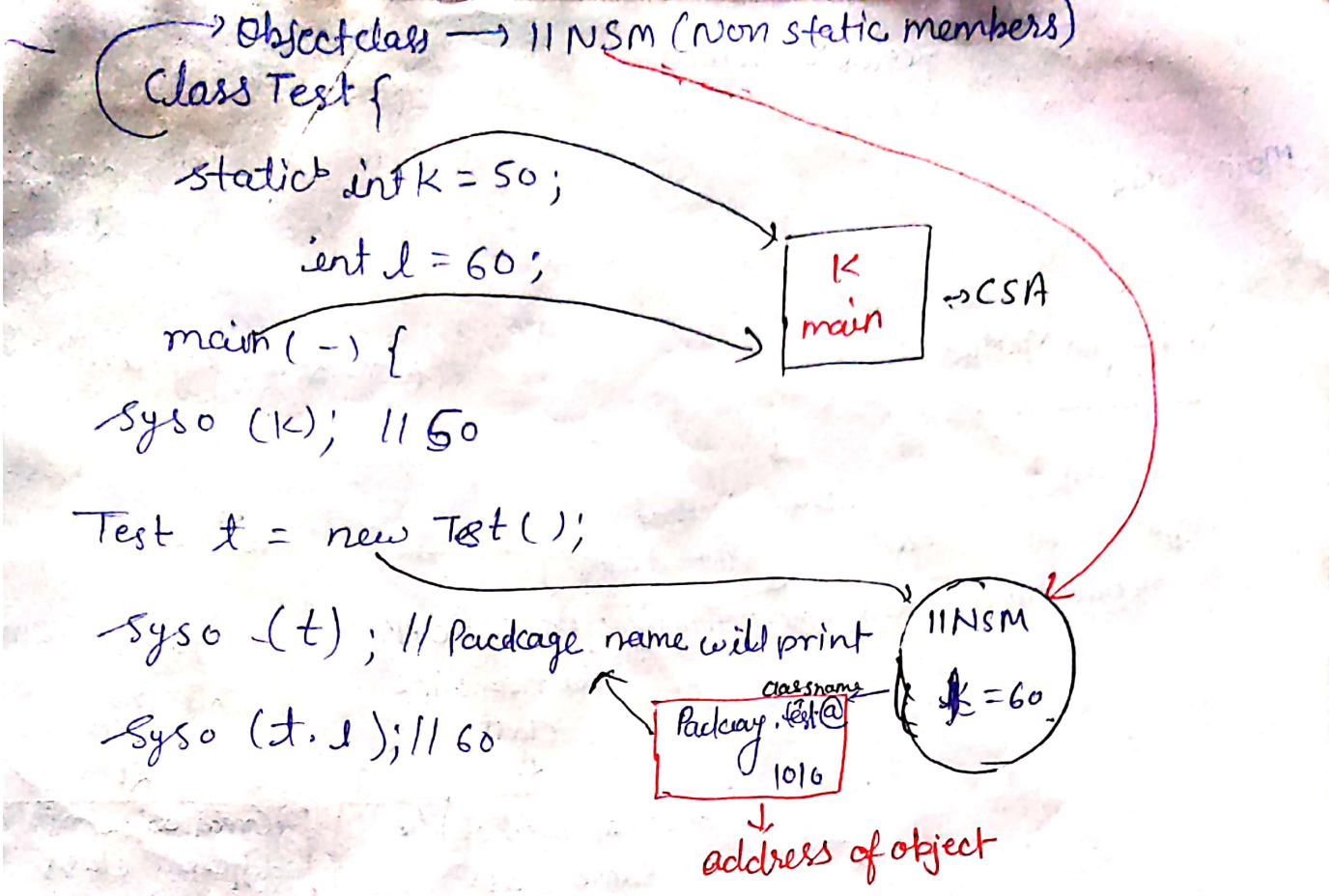
2. It will give name for empty memory block.

Q. When we can say empty memory block as object?

⇒ When all the non static members meas II NSM from the ~~class~~ object class and other from the class which we will create, will loaded in empty memory block then it will called as ~~blac~~ object

Object class - It is the supermost class in Java, which contains 11 Non-static methods (NSM). These NSM's can be loaded in all objects of Java.

Scanned with OKEN Scanner



⇒ Syntax to Create object

Classname variable name = new classname( );

constructor

Demo d = new Demo( );

Classname — N — keyword — constructor

[26/08/24]

Methods / Functions :- It is the block of code ~~with~~ which runs when we call it.

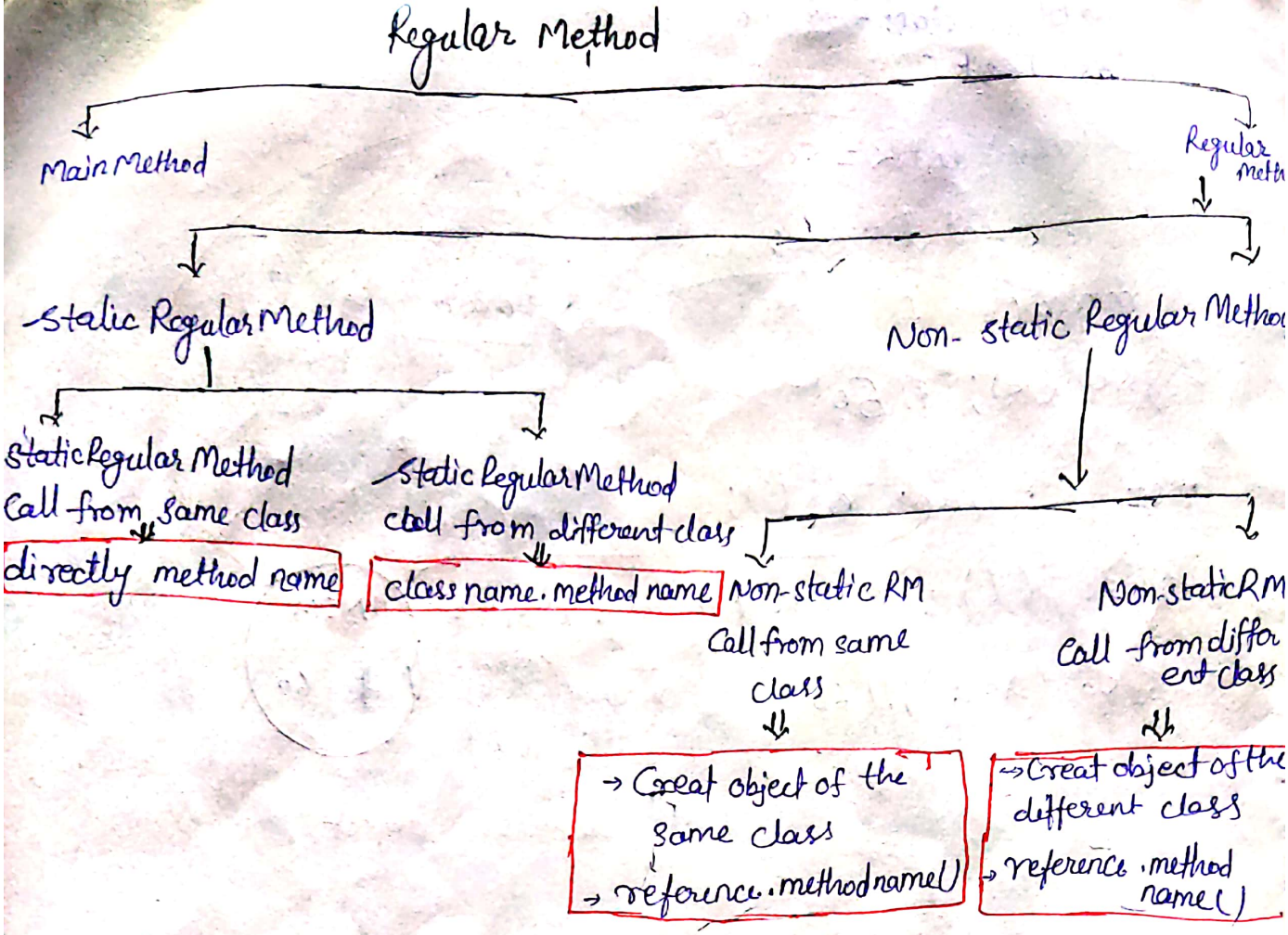
Methods

Regular method

main metho → It is responsible to run other all regular methods

Pubic static void main(--)

Scanned with OKEN Scanner



Case-1 Static Regular Method call from same class

```
Class Test { // class body open)
main(—) {// main method open
m1();
m2();
} main method body open close

Pubic static void m1() {
Syso ("running from method m1");
}
Pubic static void m2(){
Syso.("running from method m2");
}
} // class body close
```

Scanned with OKEN Scanner

User Logic class:-

A class which contains main(-) is called as user logic class.

Business Logic class:-

A class which won't contains main(-) is called as Business logic class.

Program on static regular method -from different class:-

```
Class sample { // (user logic class)
main(—) {
Sample.m1();
Sample.m2();
}
}
```

```
Class sample { // (Business logic class)
Public static void m1(){
Syso("I am from 5:30pm batch");
}
Public static void m2(){
Syso("Hi guys good evening to all");
}
}
```

Scanned with OKEN Scanner

Case3 Program on non-static method call from same class:-

```
class sample2 { // ULC (class body open)
main (-) {
sample2 s = new sample2(); // object
   s.m3 ();
   s.m4 ();
 }
 Public void m3 () {
 Syso ("I am non-static call from same class m3");
 }
 Public void m4() {
Syso(" I am NS from same class");
 }
 } // Class body close
```

Case-4 Non-static Regular method call from different class

```
Class Sample3 {
main (-) {
  Sample2 s= new sample4 ();
   s.m3 ();
   s.m4();
  }
  }
Class Sample4
Public void m5() {
Syso (" I am non static m5");
}
Public void m6() {
Syso (" I am non static m6");
} }
```

Scanned with OKEN Scanner

Assignment :-

Case-1 Static Regular Method _ Same Class

```
Package Methods;
Public class SRM_callfrom_Sameclass { // class body open
    public static void main (String [] args) { // main method body open
        m1();
        m2();
    } // main method body closed
    public static void m1(){
        Syso ("I am from Sagar M.P.");
    }
    public static void m2(){
        Syso ("Hi everyone i'm here to do selenium course");
    }
} // class body closed
```

Case-2 Static Regular Method _ different class

```
Package Methods;
Public class SRM_differentclass
    public static void main (String [] args) {
        SRM_Callfrom_Sameclass.m1();
        SRM-Callfrom_ sameclass .m2();
    }
    public static void m1(){
        Syso("I am from sagar M.P")
    }                                  } no need to write
    public static void m2(){
        Syso ("Hi everyon  ---------- ");
    }
}
```

Scanned with OKEN Scanner

[27/08/24]

Keywords & Identifiers

Keywords :- These are predefined reserved words which is having specific meaning. Each keyword performs specific task.
ex - extends, implements

Keywords are always starts with lower case

Identifiers :- These are the names given to variables, methods, classes, interface etc. But according to Java standard practices, while writing keywords we should have to follow "Rules and conventions".

Rules are mandatory, conventions as your wish.

Rules for Identifiers

→ Keywords can't be Identifiers

→ It can't accept special character/symbols except _ (underscore) and $ (dollar). ex. class Sample_ ✓ Sample# ✗

→ It must be the combination of letter and digits but it must be starts with letters ex - abc123 ✓ 123abc ✗

Conventions for Identifiers

→ While creating classes the 1st letter of classname should be in uppercase. If it is combination of multiple words each 1st letter should be uppercase/camelcase.
ex - Sample, Dimple, SampleDimple.

Scanned with OKEN Scanner

⇒ While writing variable, methods the 1st letter should be lowercase. If it is combination of multiple words then 1st letter of 1st word should be lowercase other words onwords 1st letter.

ex - sitaRam

Scanned with OKEN Scanner

Constructor :- It is the special type of method which is used to initialize non-static members of class

It will loads NSM into object which is created by new keyword.

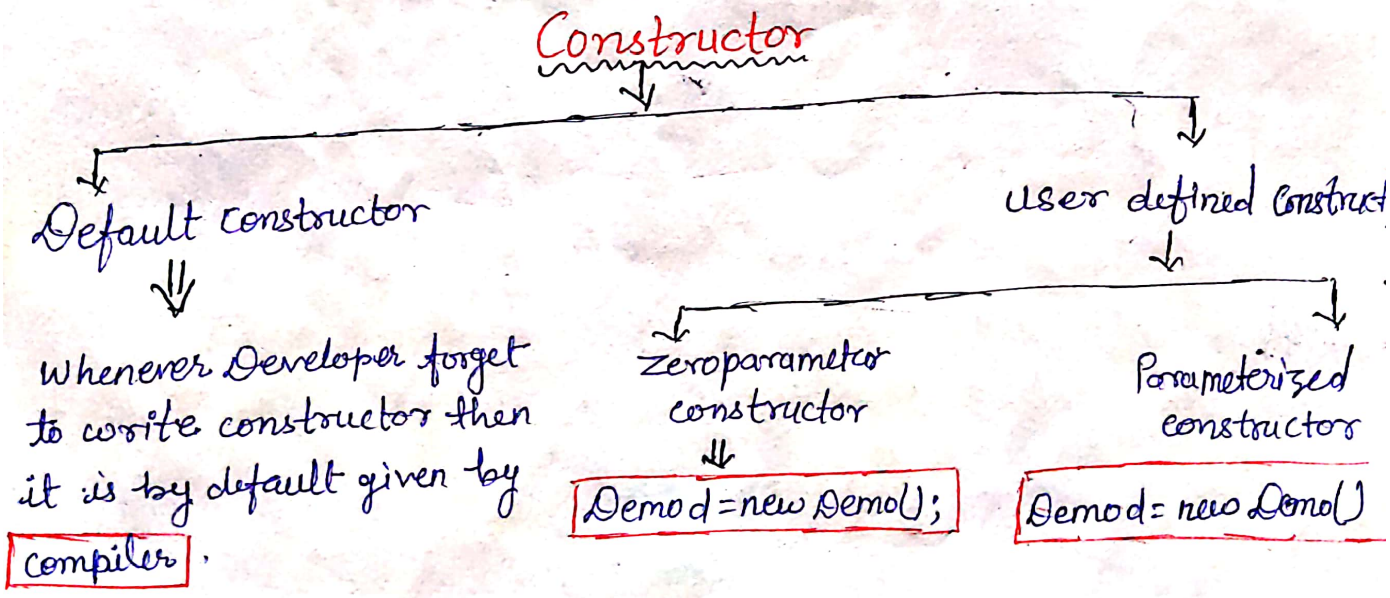
Characteristics :-

⇒ Constructor can be called during object creation automatically.

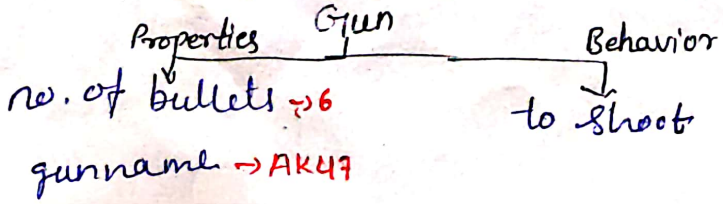
⇒ Copstructor we can call any no of times but the reference variable of object should be different

⇒ constructor does not have returtype

⇒ constructor name should be same as that of class name.



Parameterized Contructor- It will used to initialize the data ~~num~~ members during object creation.



Scanned with OKEN Scanner

```
Class Gun {
    String Gunname;
    int no of Bullets;
Public Gun (string gun name, int no of bullets) {
this. gunname = gun Name;
this. no. of bullets = no. of Bullets;
Public void shoot(){
for (int i=1; i≤ no of Bullets; i+1) {
        syso ("Deshkew");
```

```
Class shooting {
    main (--) {
    Gun g = new Gun ("AK47", 6);
        g. shoot();
    }
}
```

Output → i=1, 1≤6 no. of bullets. 6 – T – Deshkew

2≤6 → True → Deshkew (op)
3≤6 → True
4≤6 → True
5≤6 → True
6≤6 → True
7≤6 → false JVM stops.

Conditional statements:- These will executes according to the output of condition.

OR

conditional statements will executes accordingly to / based on output of condition. [It performs set of operations]

ex - if (5<8) { → condition is true.

Syso ("Hi"); → Hii will print

conditional statements are :- 1. if
2. if else
3. else if ladder
4. Switch

⇒ if - If we want to perform one set of operation then we can use if statement.

Syntax if (2<3) { → ~~Stateme~~ condition is true

Syso ("Hi"); → it will print Hi

}

⇒ if else - Whenever we want to perform 2 - set of operations then we can go through if else statement.

ex -
```
int age;
if (age >= 18) {
Syso("she is eligible for marriage");
}
else
Syso("Not eligible");
}
```

Scanned with OKEN Scanner

⇒ else if / if else ladder → When we want to perform more than 2 sets of operation we will go through else if / if else statement.

```
class student {
main (---) {
    int marks = ;
{
if (marks ≥ 65) {
    syso("Pass with 1st class with distinction");
  }
else if ( marks ≥ 55) {
    syso("Pass with 1st class");
  }
if else ( marks ≥ 45) {
    syso ("Pass with 2nd class");
  }
else if (marks ≥ 35) {
    syso ("Pass with 3rd class");
  }
else {
    syso ("Fail");
  }
  }
```

Scanned with OKEN Scanner

[31/08/2017]

⇒ Switch - Whenever case value matches with switch condition then it will execute the particular block.

Syntax -

```
Switch ("Idle")
{
Case ("Idle"){
System.out.println ("-----");
}
}
```

Program -

```
Class Hostel {
main (--) {
Switch ("IDLE") {
Case ("Dosa") {
Syso ("on monday");
break;
}
Case ("Puri") {
Syso ("on Tuesday");
break;
}
Case ("Idly") {
Sys ("on wednesday");
break;
}
```

Scanned with OKEN Scanner

```
default {
Syso ("no breakfast");
}
}
```

Scanned with OKEN Scanner

[02/08/24]

Object oriented programming (OOPS)

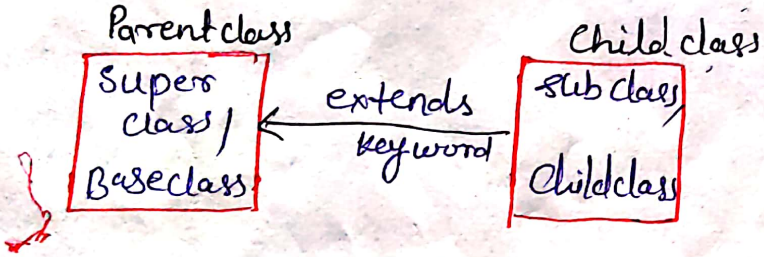
OOPS principle :-

- → Inheritence
- → Polymorphism
- → Abstraction
- Encapsulation
- Interface

} To convert real time objects, scenarios into Java program we can use "oops principles".

⇒ **Inheritence** - The process of acquiring properties from one class to another class is called "Inheritence"

It has two tools



Super class / Baseclass - The class from where we are accessing properties.

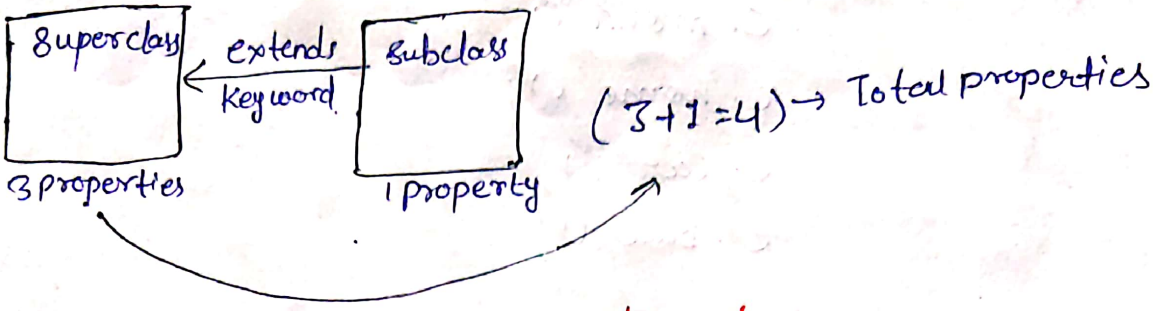
Sub class / Child class - The class in which we are properties.

Types of Inheritence - There are basicall 4 types

- → Single level Inheritence → 2 class mandatory
- Multilevel Inheritence → more than 2
- Multiple level Inheritence → more than 2
- Hirarchicle Inheritence → more than

1 super → multiple → Subclasses

Single level Inheritence → It is the type of Inheritence in which one subclass can acceqires properties from one super class.



program of single lever inheritence:

```
package singlelevel;

public class Father {
    public void home () {
    System. out . println ("2BHK");
    }
  public void money () {
    system. out. println ("50 Lakhs")
    }
  public void car () {
    system . out . println ("SKODA");
    }
    }
```

⇒ Package singlelevel

To copy or to take properties from father class extends is used

```
public class Child extends Father {
  public void bike () {
  system. out. println ("MT-15YAMAHA");
  }
  }
```

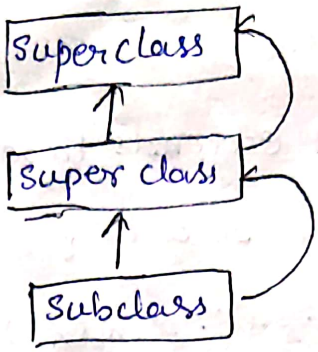
Scanned with OKEN Scanner

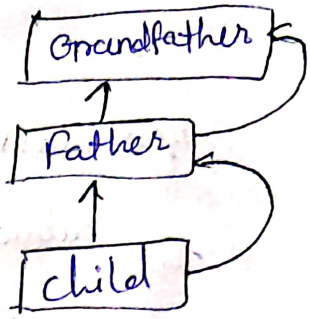
```
Package singlelevel
    class vlc
public static void main(String [] args){
    Child c = new Child ();
        c. money ();
        c. home ();
        c. car ();
        c. bike ();
    }
}
```

NOTE- Even single class in Java exhibits single level Inheritence

```
Package

public class Sample {
 public static void main (String [] args) {
    Sample s = new Sample ();
        s.
        s.
```

⇒ Multi level Inheritence - It is the type of Inheritence in which one subclass acquires the properties from other class and that class acquires properties ~~for~~ from ~~perticular~~ a perticular class





Program -

```
package multilevel

public class Grandfather {
  public void land () {
  System.out.println ("Gekars");
  }
  public void pension () {
  System.out.println ("20,000 per month");
  }
  }

public class papa extends Grandfather {
  public void car (){
    System.out.println ("CRETA");
  }
  public void money (){
    System.out.println ("50 Lakhs");
  }
```

Scanned with OKEN Scanner

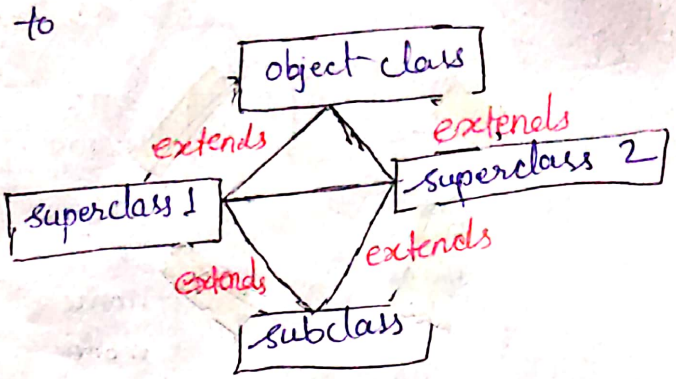
```
public void home() {
    System.out.println("2 BHK Bungloo");
}
}

public class Son extends Papa {
    public void bike () {
        System.out.println ("MT-15 YAMAHA");
    }
}

public class MultilevelInheritence {
    public static void main (String [] args) {
        Son S = new Son();
        S.land();
        S.pension();
        S.money();
        S.home();
        S.car();
        S.bike();
    }
}
```

Scanned with OKEN Scanner

Multiple ~~level~~ Inheritence

Having 2 superclass and one subclass

In this, one subclass trying to acquire properties from two superclass.

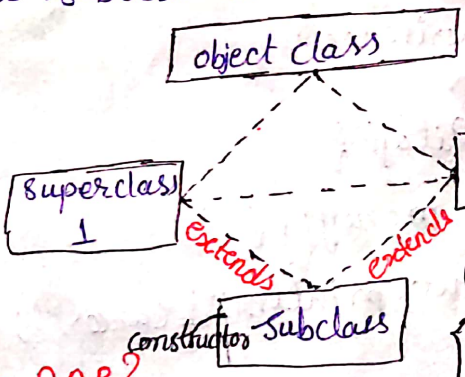


→ This is called as "Dimond Ambiguity Problem" / constructor chaining problem.

[03/09/21]

How DAP accures?

The ~~superclass~~ supercall statement of subclass constructor is trying to fetch properties from 2- super classes. at the same time. So ~~cb~~ subclass will have confusion, on that time DAP occurs



How we can solve DAP?

⇒ using Interface

Can Java supports multiple Inheritence using class?

→ No

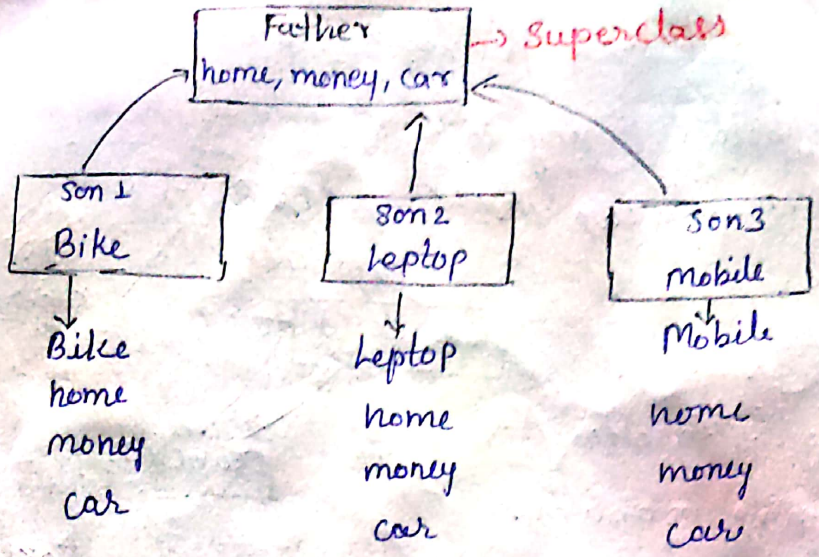
Can Java supports multiple Inheritence using Interface?

→ Yes

⇒ Hierarchical Inheritence

It is the type of Inheritence in which multiple subclasses are acquiring properties from single super class.

Scanned with OKEN Scanner



Program :-

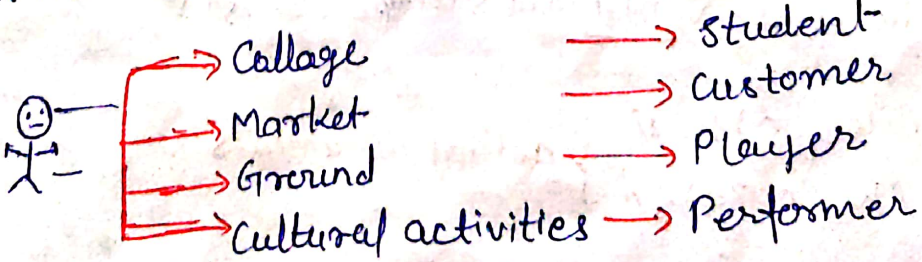
```
package Inheritence;

public class Pitaji {
public void money() {
  System.out.println("1 Carror");
}
public void car() {
  System.out.println ("SKODA")
}
public void home() {
  System.out.println ("3BHK");
}
}
public class Son extends Pitaji {
public void Laptop () {
System.out.println ("ASUS")
```

```
Public class Son2 extends Pitaji {
    public void Bike () {
        System.out.println("KTM");
    }
}
public class Girl extends Pitaji {
    public void Mobile (){
        System.out.println("MOTOROLA");
    }
}
public class Heirarchy {
    public static void main (String[] args) {
        Son s = new Son();
        s.home();
        s.car();
        s.money();
        s.laptop();

        Girl g = new Girl();
        g.car();
        g.home();
        g.money();
        g.Mobile();

        Son2 e = new Son2();
        e.home();
        e.car();
        e.money();
        e.Bike
    }
}
```

⇒ Poly

⇒ **Polymorphism** → It is the principle / concept of oop; in which one object can shows multiple behaviors. ~~is~~

Poly morphism is a latin word ~~which~~ Poly means many and morphism means Behaviors (Forms).

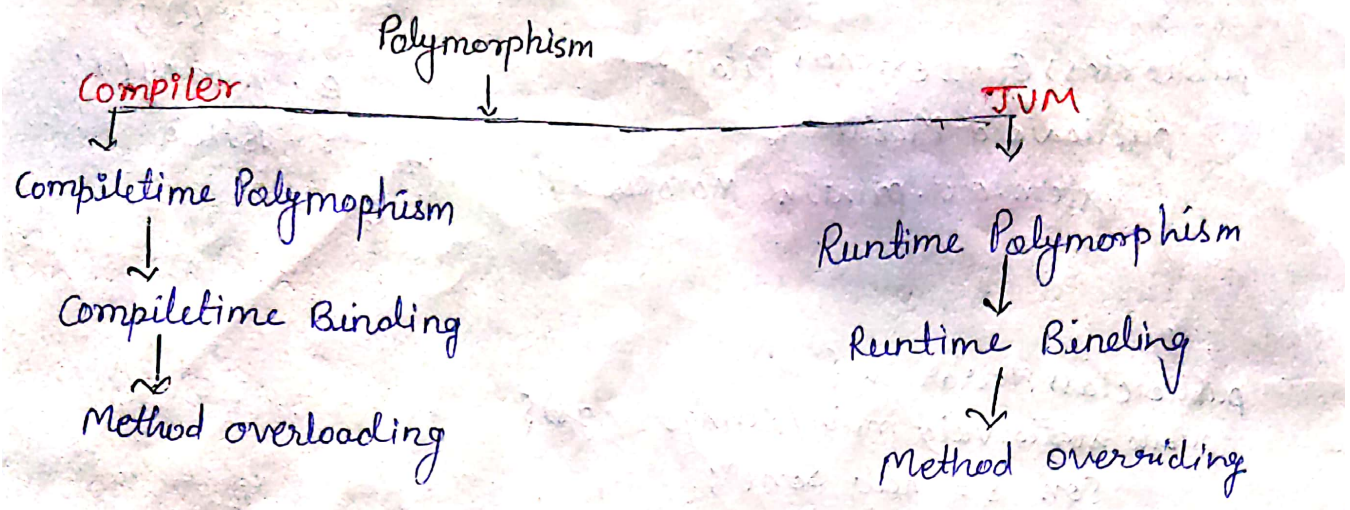


Scanned with OKEN Scanner

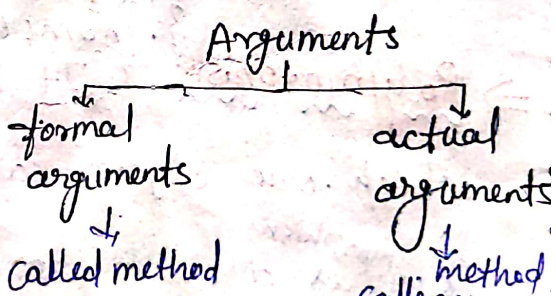
There are two types of Polymorphism 04/09/24

(i) Compiletime Polymorphism → Example → method overloading

(ii) Runtime Polymorphism → Example → method overriding



Compiletime Binding :- The process of matching actual arguments of method calling statement with respect to formal arguments of called ~~statement~~ method is called as CTB :




```
Class A {
  public static void main(String[] args) {
    add(10,20);  //
    add(10,20,30);//
  }
  Public static void add(int a, int b) { //
      int c=a+b;
    System.out.println(c);
  Public static void add(int a, int b, int c){//
      int z = a+b+c
```

} actual arguments

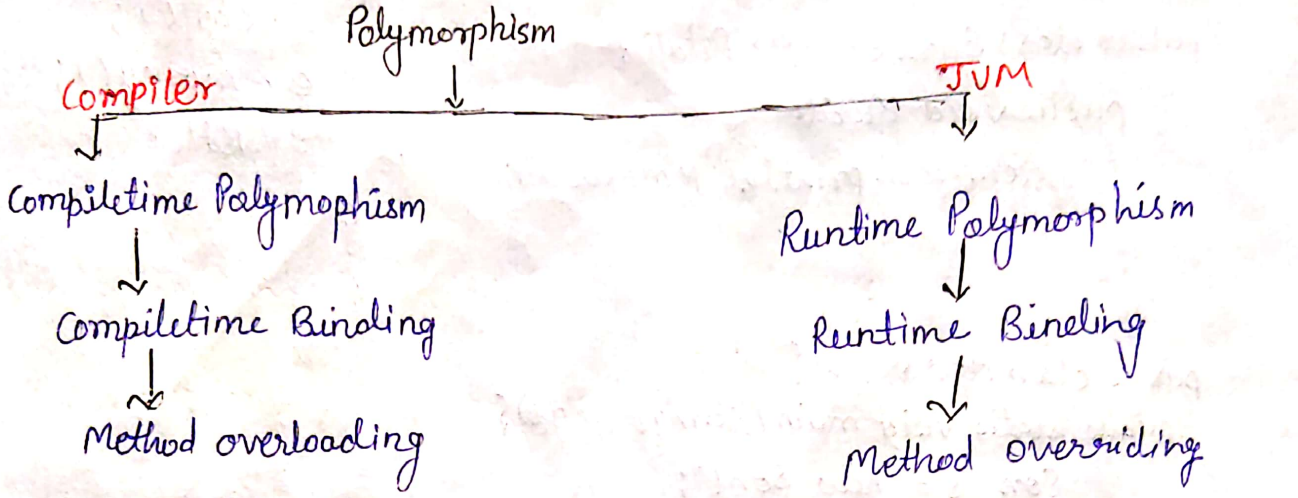
} formal argument

Scanned with OKEN Scanner

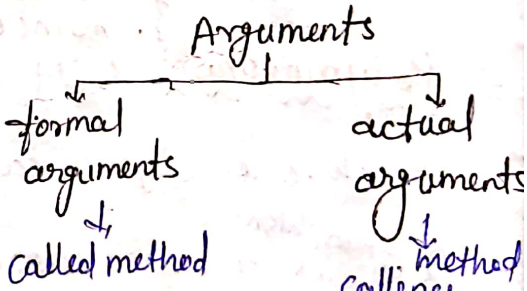
There are two types of Polymorphism. 04/09/25

(i) Compiletime Polymorphism → Example → method overloading

(ii) Runtime Polymorphism → Example → method overriding



Compiletime Binding :- The process of matching actual arguments of method calling statement with respect to formal arguments of called ~~statement~~ method is called as CTB:




```
Class A {
  public static void main(String[] args) {

add(10,20); //
add(10,20,30); //
  }
Public static void add(int a, int b) { //
        int c=a+b;
 System.out.println(c);
Public static void add(int a, int b, int c){//
        int z = a+b+c
 System.out.println(z);
```

} actual arguments

} formal argument

Scanned with OKEN Scanner

Method overloading :- Within the class multiple methods is having same method names but different arguments /parameters is called as "method overloading"

General scenarios :- → ATM
→ switch off Button of phn → on, off, reboot
→ human being → sc, end call

NOTE - Every static method is overloaded.

[We can overload constructor]

Runtime Polymorphism :-

```
Package

class Gun

Public static void
  string gunname;
  int no. of Bullets;
public Gun(string gunnam, int no. of Bullets)
  this.gunname = gunname();
  this.no of Bullets = no. of Bullets();
  }
Public void shoot(){
  for (int i=1, i≤ no. of Bullets, int++);
    System.out.println("Deshkew")
  {
  }
```

Scanned with OKEN Scanner

```
class shooting {
main ( -- )
Gun g = new Gun ("AK47", 6);
}
}
```

Runtime Binding:- The process of matching actual and formal arguments during object creating is called as "RTB". It is done by "JVM."

Method overriding:- The process in which method declaration is same but method implementation is changing or overriding according to subclass specification is called as "Method overriding." It is the example for Runtime polymorphism.

```
Package RTB
Class Animal {
public void animalstalk () { Method declaration
System.out.println (" Animals are talking");
}
}
Class Cat Extends Animal
public void animals talk () {
System.out.println ("Meow, Meow");
}
}
```

Scanned with OKEN Scanner

```
class ULC {
public void static void main[String[] args]
    Animal a = new cat();
    a. Animals talk();
    cat c = new cat();
    c. animalstalk();
}
}
```

Method overriding

Difference b/w Compiletime polymorphism and Runtime polymorphism.

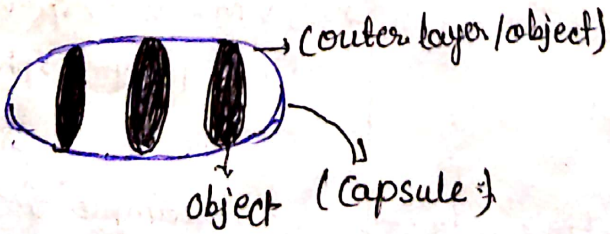
Compiletime polymorphism	Runtime polymorphism
→ During compiletime it can be done	→ During ~~Compile~~ Runtime / executio of program it can be done.
→ The binding which is done during compiletime polymorphism is called as Early Binding / Static Binding / CTB.	→ The binding which is done during Runtime is called as late Binding dynamic Binding / RTB.
→ Compiler is responsible for compile time polymorphism.	→ JVM is responsible for Run-time polymorphism
→ Example for compiletime polymor-phism is over loading.	→ Example for Runtime polymor-phism is overriding.

Scanned with OKEN Scanner

⇒ Difference b/w Method overloading & Method overriding

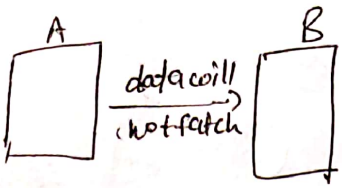
Method Overloading	Method Overriding
→ Method overloading is having same method names but different parameters / arguments.	→ In method overriding, method declaration is same but method implementation is changing according to subclass specification.
→ All static methods overloads	→ All non-static methods override
→ In method overloading inheritence is not require.	→ Method overriding require inheritence.
→ Object creation is not necesary	→ object creation is necessary
→ It is the example of compile-time polymorphism	→ It is the example of Runtime polymorphism.

Encapsulation:- The process of wrapping in which one object is warapped inside another object is called as "Encapsulation".

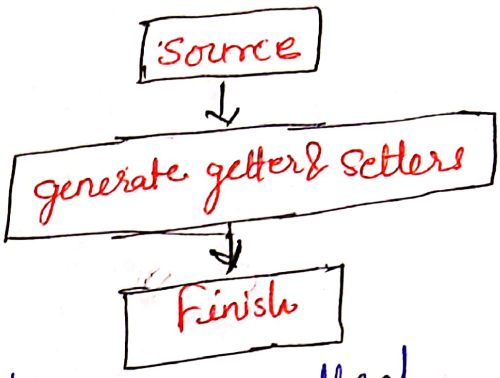


Scanned with OKEN Scanner

NOTE - ⇒ The data members of Encapsulated class is always private



⇒ We can access even private data members of Encapsulated class out side the using getters and setters.



⇒ Abstraction:-

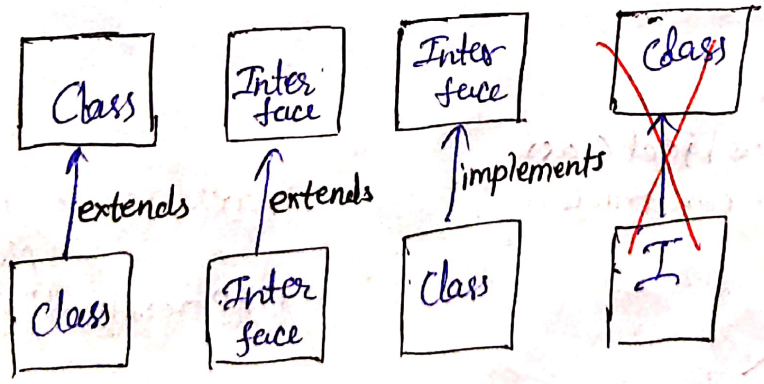
The process of hiding Internal implementation and providing only functionality to end user is called as "Abstraction".

⇒ Can we achieve 100% Abstraction using Abstract class?

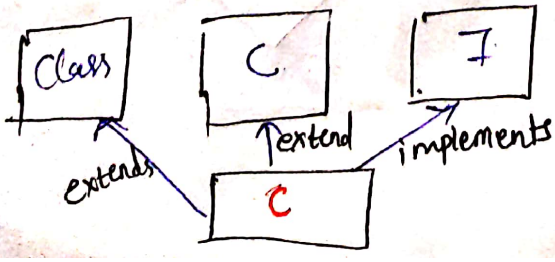
⇒ NO

⇒ How we can achieve 100% abstraction in Java?

⇒ Using Interface.



Same class → extends
different → implement



Scanned with OKEN Scanner

Class

→ we have constructor

we can create object of constructor

object class is the supermost class so it results in DAP

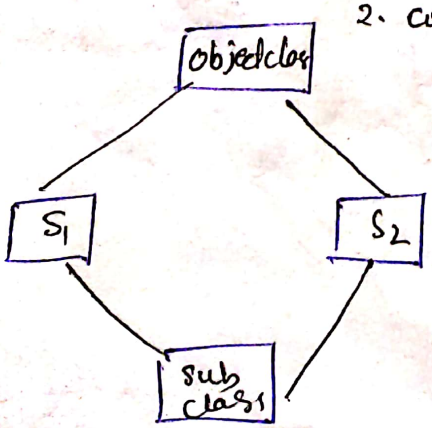
Interface

→ we don't have constructor

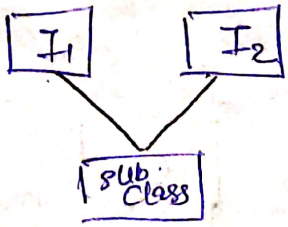
→ we can't create object

→ Object is not supermost class for Interface so it removes DAP.

DAP → 1. object class
2. constructors

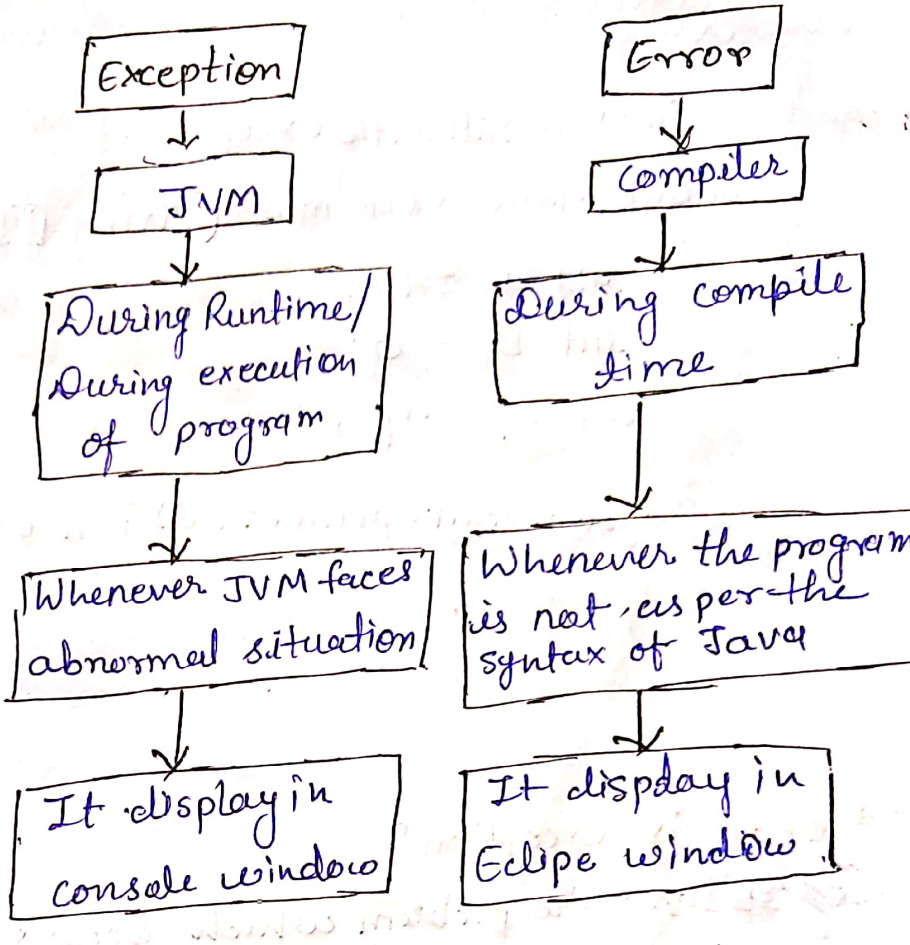


Interface → No objectcla
→ No constructo
" DAP resolved "



Scanned with OKEN Scanner

Exception :-



⇒ The proplem which occurs during runtime is called as "Exception."

OR

⇒ The unexpected event which can't handle by JVM without try and catch blocks is called as "Exception".

* All exception will be displayed in console.
* All exceptions are present inside "Java.lang.package".

Scanned with OKEN Scanner

Program →

```
Class ArithmeticException{
public static void main(String[] args){
    int a =10;
    int b = 0;
    int c = a/b;
    System.out.println(c); // Exception (ArithmeticExcepti
    }
    }
```

→ What is Exception?

- ~~Exception~~ :- The problem which occurs during runtime is called as "Exception."

→ Exception are always created by JVM, which stops normal flow of the program.

Types of Exceptions:-

⇒ ArithmeticException
⇒ Nullpointer Exception
⇒ Number formate Exception
⇒ String Index out of Bounds Exception
⇒ Array Index out of Bounds Exception
⇒ Class cast Exception

⇒ Arithmatic Exception. If any digit (expect zero) devided by zero then it results "Arithmatic exception."

Scanned with OKEN Scanner

Program -

```
public Class Division{
public static void main (String[] args){
        int a = 10;
        int b = 0;
        int c = a/b
System.out.println (c)
}
}
```

⇒ Null pointer Exception :- If the refrence of object is null then it results in "Null pointer exception."

✗ Demo d = new Demo();

Demo d = null;

```
Class NPE {
 int i =10;
public. static void main (string[] args){
   NPE. e = null;
System.out. println ("e.i) ; // Null pointer Exception.
}
}
```

Exception in thread "main" cannot read

⇒ Number formate Exception :- If we give any single character as an argument to parse int method then it results in "Number formate Exception".

Program -

```
public Class NFE {
static public static.
    System.out.println(Integer.parseInt("20k4"));
}
}
```

⇒ Character are not allowed in parse Int method.

Q→ Why wrapper classes are needed ?

⇒ To create Java as 100% object oriented.

⇒ String Index out of Bounds Exception :-

When Index is not present but still we are trying to fatch data is called "String Index out of Bounds Exception."

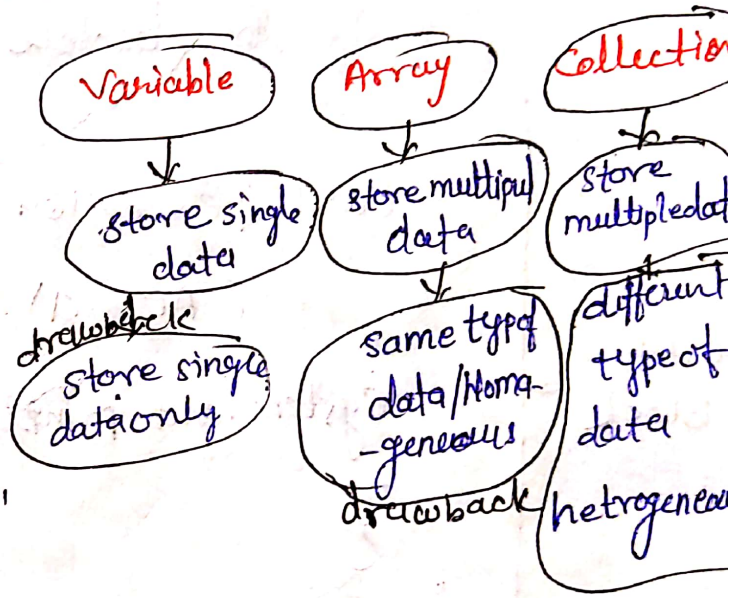
Scanned with OKEN Scanner

Program :-

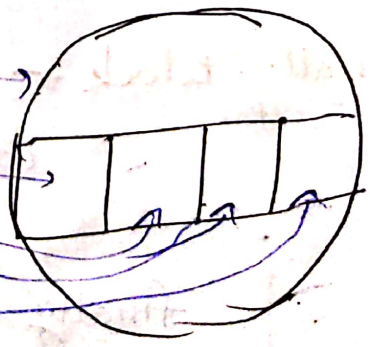
```
public class SIOOBE {
public
    string s = "DEMO";
system . out . println (s . charAt (5));
}
}
```

⇒ Array Index out of Bounds Exception :-

When Index is not present but still we are trying to fatch data is called "Array Index out of Bounds Exception."




```
String [] s = new string [4];
    S[0] = "Anand";
    S[1] = "Bablu";
    S[2] = "Chandu";
    S[3] = "Dhanush";
System.out.println (S[5]); // AIOOBE
```



Scanned with OKEN Scanner

[10/09/24]

Handling Exception – We can handle exception using try and catch blocks.

Try block :- It is the block in which we can write risky code.

```
int a=10;
int b=0;
int c = a/b   → risky code
```

output $10/0 = \infty$

⇒ **Catch block :-** Each catch block is followed by try block.

Catch block ⇐

```
Catch (Exception name referencevariable) {
    (AE. a) {
System.out.println("Exception is Handle");
}
```

⇒ **finally block :-** It will execute irrespective try and catch block.

```
finally {
    System.out.println("Thanks you for using SBI ATM");
}
}
```

Scanned with OKEN Scanner

Exception

```
class A {
public static void main(String[]args){
    int a =10;
    int b =0;
    int c = a/b
System.out.println(c); //AE
    }
}
```

Exception Handling

```
class EHandling {
public static void main(String[]args){
int a =10;
int b = 0;
try {
  int c = a/b
  }
catch (Arithmatic Exception e) {
System.out.println("Exception is Handle");
  }
}
```

program for Arithmatic Exception Handling

```
class Division{
public static void main (String[] args){
    int a = 30;
    int b = 0;
try{
  int c = a/b
  }
 Catch (ArithmaticException e) {
System.out.println("Exception is Handled");
}
}
```

Scanned with OKEN Scanner

Program for ...

```
public class NPE_Handling {
        int i = 20;
        int h = 10;
public static void main (String [] args) {
    try {
    NPE_Handling n = null;
    System.out.println (n.i);
    System.out.println (n.h);
    }
    Catch (NullPointerException e) {
     system.out.println ("Exception is handled");
     }
    }
}
```

Program for Number Format Exception Handling

```
public class NFE_Handling {
public static void main (String [] args) {
 try {
  system.out.println (Integer.parseInt ("2K24"));
 }
 Catch (NumberformatException m) {
   system.out.println ("Exception is handled");
 }
}}
```

Scanned with OKEN Scanner

Program for String Index Out of Bounds Exception Handling.

```
public class SIOOBE_Handling {
public static void main (String [] args) {
    String s = "ALLEN";
System.out.println(s.charAt(4));
  try {
    system.out.println(s.charAt(6));
  }
catch (StringIndexOutOfBoundsException d) {
    system.out.println("Exception is handled");
}
}
}
```

Program for Array Index

```
public class AIOOBE_Handling {

    public static void main (String [] args) {
        String [] s = new String [5];

            s[0]  = "POOJA";

            s[1]  = "RADHA";

            s[2]  = "SIYA";

            s[3]  = "TANVI";

            s[4]  = "VIBHA";

    System.out.println (s[1]);

    System.out.println (s[3]);

     try {
     System.out.println (s[7]);
      }
     catch (ArrayIndexOutOfBoundsException v) {
        System.out.println("Exception is handled");
     }

     }

     }
```

Scanned with OKEN Scanner

Interface :- It is the oops principle which is 100% abstract in nature. The methods inside the Interface are public & abstracts.

```
void m1();
void m2();
```

* The variable inside the Interface are static & final.
* It won't contains constructor
* We won't create object for Inte
* It resolve DAP.
* We don't have object as Supermost class for Interface.
* Using implementation class we can provide body for methods which are present in Interface.

*NOTE → To connect Interface & implementation class we can prefer [implements keyword].

program for Interface :-

```
Interface Demo{
    void m1();
    void m2();
}
```

Implementation class

```
class Sample implements Demo{
public void m1(){
System.out.println("I am m1 completed from IC");
}
Public void m2(){
System.out.println("I am m2 completed from IC");
}
}
```

Scanned with OKEN Scanner

```
Class ULC {
public static void main (String [] args) {
  Sample  S = new Sample ();
        S. m1 ();
        s . m2 ();
System.out.println ()
    }
    }
```

Output - I am m1 completed from IC

I am m2 completed from IC

Program :-

```
public interface A {

    void I1();
    void I2();
  }

public class B implements A {
   public void I1 () {
      System.out.println ("I am a natorious girl");
   }
public void I2 () {
      System.out.println ("I will go outside on saturday");
  }
   }
```

Scanned with OKEN Scanner

```
public class C {
    public static void main(String[] args) {
        B  I = new B();
        I.I1();
        I.I2();
    }
}
```

[11/09/24]

String :-

→ String is non-primitive data type
→ String is collection of characters
→ String is inbuilt class which is present java.lang.pkg
→ String objects we can create in 2 way
① Using new keyword `String s = new String("Demo");`
② without using new keyword `String S = "Ankush Kumar";`

String function :-

⇒ **Length :-** It is used to fetch the no. of characters in string.

"NARESH IT"

`System.out.println(s1.length());` o/p = 8

Scanned with OKEN Scanner

Program for String length function

```
public class strings {
public static void main (string [] args) {
        S1 = "NARESHIT";
        S2 = "nareshit";
        S3 = " NARESH";
System.out.println (S1.length());
}
}
```

=> to uppercase(); → It converts lowercas string to uppercase string

```
S1 = "NARESHIT";
S2 = "nareshit";
System.out.println (S2.touppercase()); O/P "NARESHIT"
```

Program for string to uppercase();

```
public class string {
public static void main (string [] args) {
        S1 = "NARESHIT";
        S2 = "nareshit";
        S3 = "NARESH";
```

Scanned with OKEN Scanner

```
System.out.println(s2.toUppercase());
```

O/P: NAREHIT

```
    }
}
```

toLowercase(); – It converts uppercase String to lowercase string

```
public class string {
public static void main (String[] args) {
    String s1 = "NRESHIT";
    String s2 = "nareshit";
    String s3 = "NARESH.IT";
system.out.println(s3.to lowercase());
system.out.println(s1.tolowercase());
    }
}
```

equalIgnoreCase();- It compare strings without caring about cases.

```
String s1 = "NARESHIT";
String s2 = "nareshit";
system.out.println(s1.equalIgnoreCase(s2));
```

O/p = true

Program

```
public class String_methods {
public string_methods
String s1 = "NARESHIT";
String s2 = "nareshit";
system.out.println(s1.equalsIgnoreCase(s2));
}
}
```

Scanned with OKEN Scanner

contains():- It checks whether particular string is present in given String or not.

```
S1 = "NARESHIT";
S2 = "nareit";
S3 = "NARESH";

System.out.println(S2.contains());
```

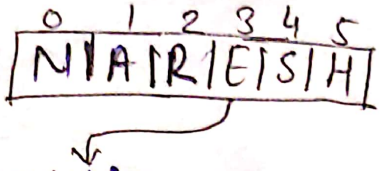
Concat():- It is used to connect two Strings.

```
S1 = "NARESH
```

Scanned with OKEN Scanner

⇒ charAt():- It will give character of particular index.

S_3 = "NARESH";



System.out.println(S3.charAT(3)); O/P = E

Program

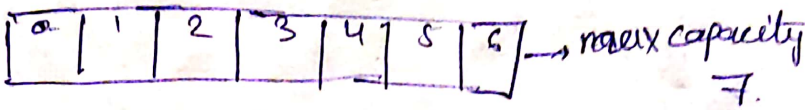
⇒ indexOf();- It will give index of particular charecter.

⇒ replace():- It will replace String by particular value (it should also be a string)

Scanned with OKEN Scanner

Array :- It is used to store multiple data's of similar (Homogeneous) types.

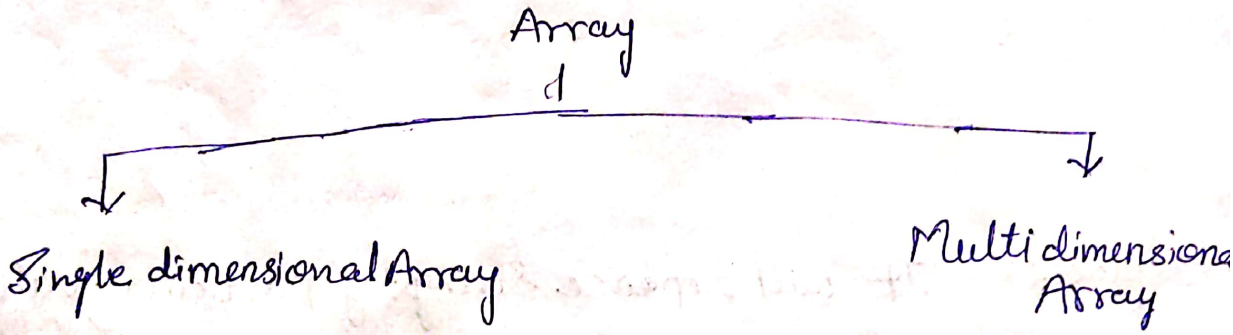
Array can stores data in the form of indexs



Array Index always starts from zero

Array Syntax:-

```
Datatype [] Array name = new Datatype [ ];
                                  └→ Array Capacity
```



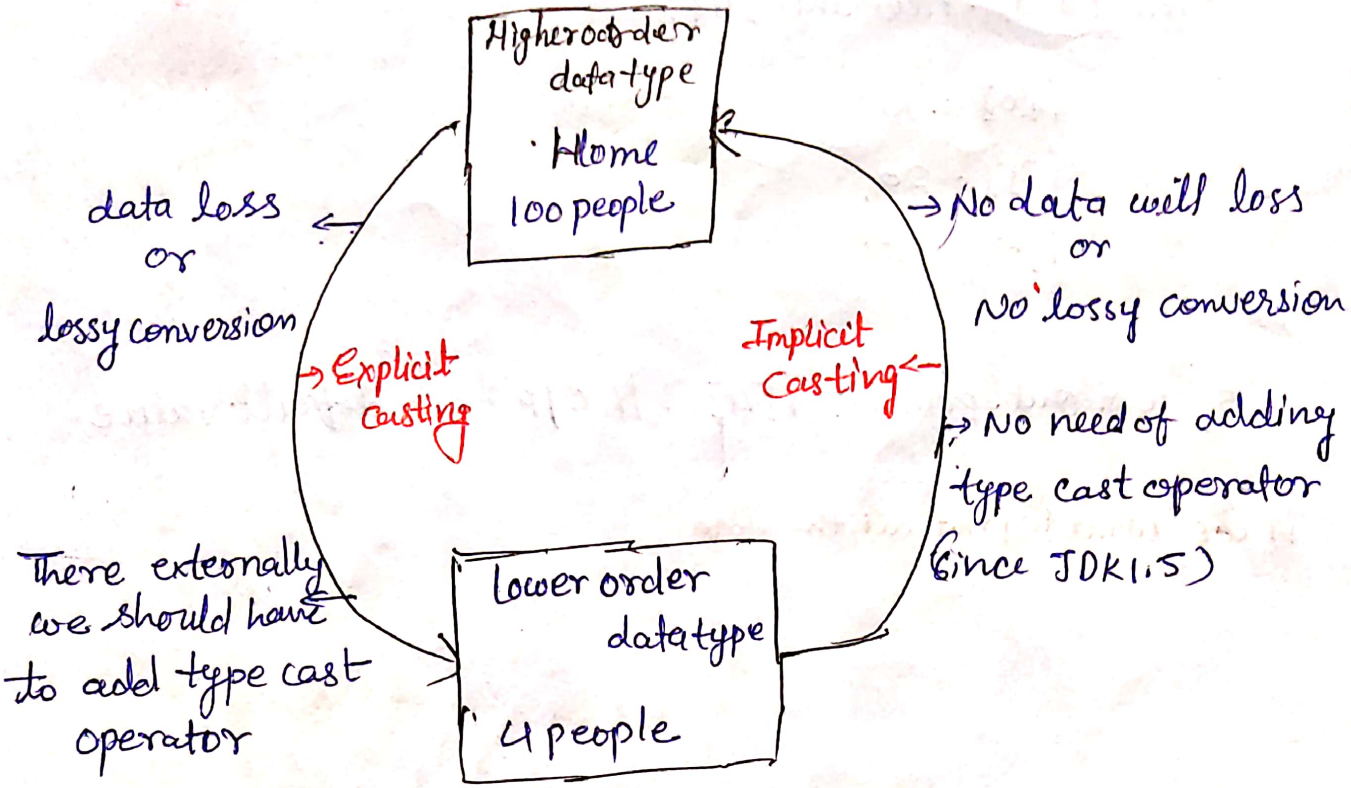
```
public class IntArray
Public static
    int [] i = new int [4];
        i[0] = 10;
        i[1] = 20;
        i[2] = 30;

System.out.println(i[4]); // O/P = 0 default value
```

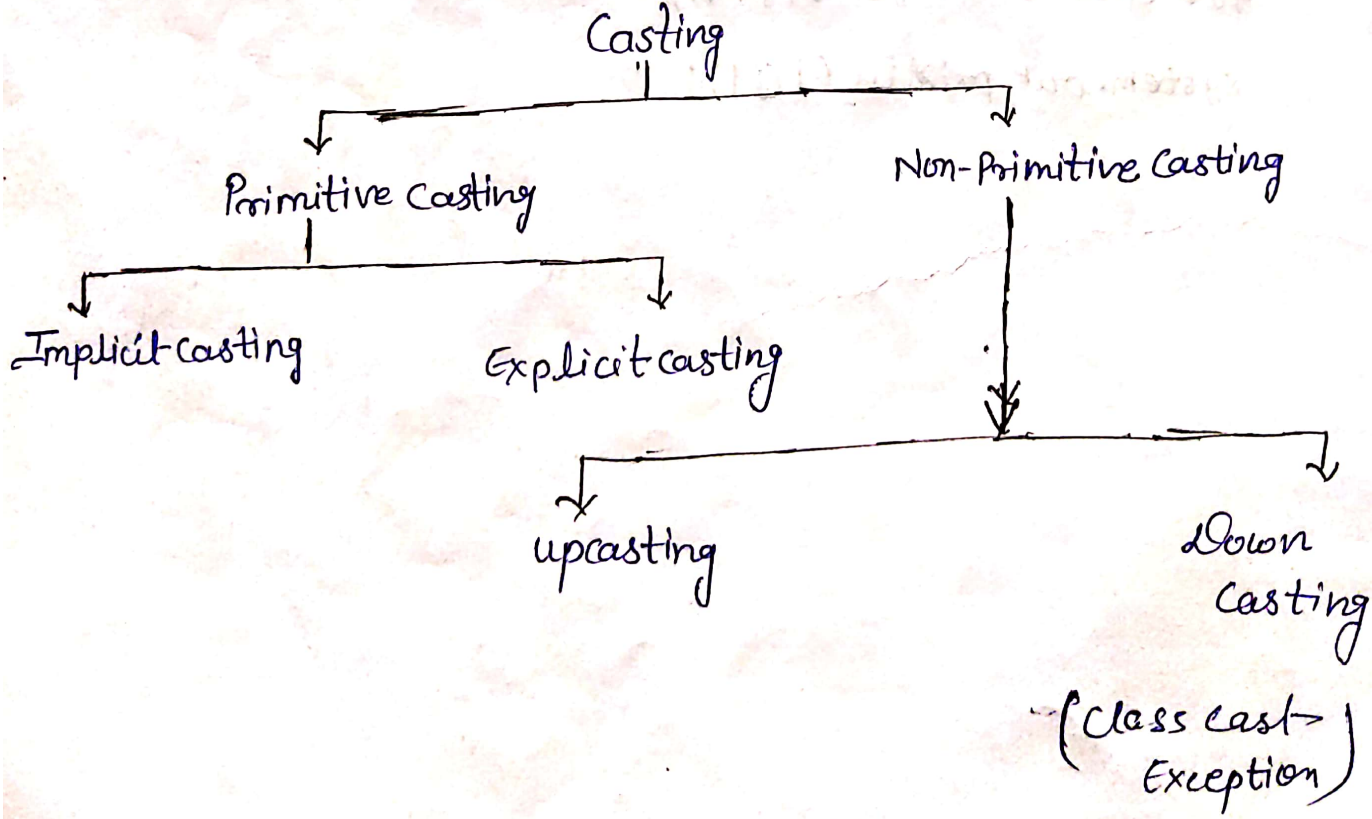
if we want to print all the data

```
for (int j=0; j<=3; j++);
System.out.println (i[j]);
```

Scanned with OKEN Scanner

Casting :- The process of converting one data type to another datatypes





Implicit Casting :- The process of converting lower order data type to higher order data type is called as "Implicit casting"

It is done automatically from JDK1.5 onwards

There is no loss of data (lossy convertion is not there)

```
double d = 10;
```

- 10 → This integer data is there
- 10 → integer type data
- int (4 byte)
- double d (8 byte) (decimal)
- integer type data is automatically converted into decimal by compiler

```
public class Implicat{
public static void main (String[] args){
    double d = 10;
    system.out.println(d); // O/P → 10.0
}
}
```

Explicit Casting :- The process of converting Higher order data type to lower order data type is called as "Explicit casting."

→ There is loss of data means lossy convertion is there

→ There exernally we should add type cast operator.

```
int i = (int) 10.8;
```

- i → integer type data
- (int) → Type Cast operator
- 10.8 → double data

Scanned with OKEN Scanner

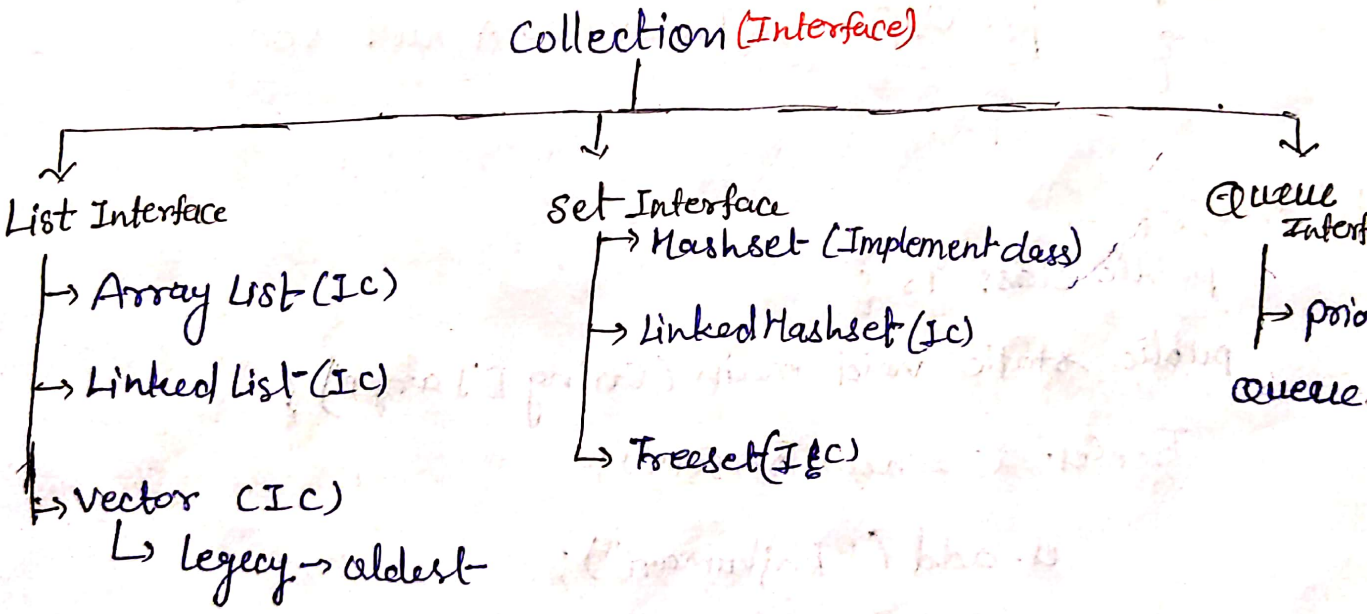
```
public class Explicit {
public static void main (String [] args) {
    int (i) = (int) 10.8;
System.out.println(i); // O/P = 10
}
}
```

[13/09/24]

Scanned with OKEN Scanner

→ Collection is the readymade framework and It acts as Interface also. which is subdevided into multiple SubInterfaces.

→ It stores multiple datas [heterogeneous]
→ The size of collection is growable.
→ It removes disadvatages of Arrays.



Scanned with OKEN Scanner

```
public class AL{
public static void main (String[] args){
    Arraylist al = new Arraylist();
        al.add ("Rajkumar");
        al.add (500);
        al.add ('A');
        al.add (null);
        al.add (500); // Duplicates are allowed
System.out.println(al);
}
}
```

// O/P → Rajkumar, 500, A, null, 500

Treeset :-

```
public class TS {
public static void main (String [] args) {
Treeset t = new Treeset();
        t.add ("Rajkumari");
        t.add ( 500 );
        t.add ('A')
System.out.println (t);
}
}
```

O/P → class casting Exception

Scanned with OKEN Scanner

Treeset :-

It won't allow heterogeneous data

It allows digits but stores in the form of ascending order.

```
public class Ts {
public static void main (String [] args) {
  TreeSet t = new TreeSet ();
    t.add (400);
    t.add (500);
    t.add (100);
    t.add (300);
    t.add (200);
System.out.println (t);
  }
}
```

O/P → 100, 200, 300, 400, 500

Hashset :- To remove Duplicates

ex.

```
Hashset hs = new Hashset();
    hs.add (100);
    hs.add (200);
    hs.add (200);
    hs.add (300);
System.out.println (hs);
  }
```

O/P → 100, 200, 300

Scanned with OKEN Scanner

Program:-

```
public class HS {
public static void main(String[] args){
HashSet h = new HashSet();
    h.add(300);
    h.add(500);
    h.add(800);
    h.add(500);
    h.add(600);
System.out.println(h);
}
}
```

O/P → 300, 500, 600, 800

⇒ Where we can use collection in Selenium?

① findElements() :→ It finds multiple methods.
The return type of this method is

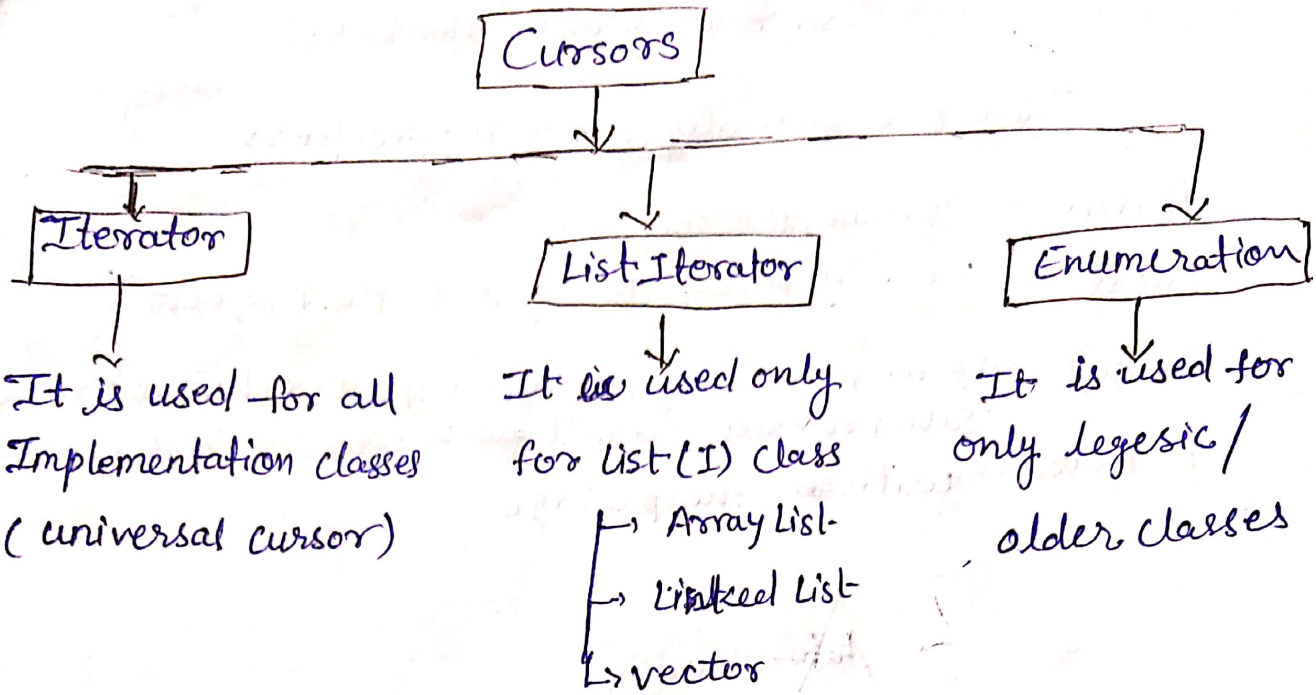
List <web Interface>
↳ Interface in collection

② getWindowHandles() :→ It stores all windows.
The return type of this method is

Set <String>
↳ Collection

Scanned with OKEN Scanner

Cursors :- These are used to fetch data from collections




```
public class HS{
```

Scanned with OKEN Scanner

[4/09/24]

Access Specifiers :- These are used in Java to decide the scope of global members.

There are basically 4 access specifiers

Privite → within the class

public → we can access throughout the project

default → If we forget to write access specifier then internally Java provides default access specifier [within the pack

protected → within the package

Main method :-

main (-) is used to run the regular methods.

Syntex :-

```
public static Returntype(since it won't return anything) void main(String [] args) {
```

public → access specifier

static → Keyword

void → Returntype

main → Method name

(String [] args) → Command line argument

⇒ Why main (-) is always public ?

Because we can access anywhere so always it is public.

Scanned with OKEN Scanner

⇒ Why it is static?

⇒ Because we can call it without creating any object.

⇒ Why it is void?

⇒ Because it won't return anything.

⇒ Can we make any changes in the sytax of main(-)?

Yes.

→ public static void main (String[] args)

→ static public void main (String [] args)

→ public static void main (String args [])

→ public static void main (String [] Rajkumar)

Scanned with OKEN Scanner